IDW
AF

ATTORNEY DOCKET NO.: EMC01-11(01046)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Richard Francis Cormier, Andrew Bruce and Svetlana Patsenker
Application No.:	09/967,111
Title:	METHODS AND APPARATUS FOR MANAGING PLUG-IN SERVICES
Filed:	September 28, 2001
Examiner:	Thanh T. Ha
Group Art Unit:	2194
Conf. No.:	8094

Certificate of Mailing Under 37 C.F.R. §1.8

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: **MAIL STOP APPEAL BRIEF PATENTS**, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on:

Date: February 8, 2006 By: Farah Z. Frasco
(Typed or printed name of person mailing Document, whose signature appears below)

Signature: _____

MAIL STOP APPEAL BRIEF PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL LETTER

Sir:

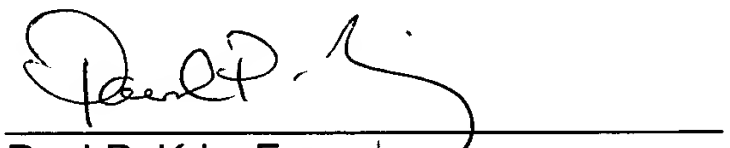
Enclosed is/are:

[x] Transmittal Letter (this form, 2 pages, in duplicate), Total Pages: 4;
[x] Appeal Brief, Total Pages: 37;
[x] Return Receipt Pre-paid Postcard (in duplicate), Total postcards: 2;
[x] Authorization to charge Deposit Account No. 50-3735, if necessary;
[x] Check in the amount of: $500.00 to cover the cost of the Appeal Brief.

Applicant hereby petitions for any extension of time which is required to maintain the pendency of this case. If there is a fee occasioned by this response, including an extension fee, that is not covered by an enclosed check, please charge any deficiency to Deposit Account No. 50-3735.

If the enclosed papers or fees are considered incomplete, the Mail Room and/or the Application Branch is respectfully requested to contact the undersigned collect at (508) 616-9660, in Westborough, Massachusetts.

Respectfully submitted,



Paul P. Kriz, Esq.
Attorney for Applicant
USPTO Registration No.: 45,752
Chapin Intellectual Property Law, LLC
Westborough Office Park
1700 West Park Drive
Westborough, Massachusetts 01581
Telephone: (508) 616-9660
Facsimile: (508) 616-9661

Attorney Docket No.: EMC01-11(01046)

Dated: February 8, 2006

Attorney Docket No.: EMC01-11(01046)
LARGE ENTITY

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Richard Francis Cormier, Andrew Bruce and Svetlana Patsenker
Application No.:	09/967,111
Title:	METHODS AND APPARATUS FOR MANAGING PLUG-IN SERVICES
Filed:	September 28, 2001
Examiner:	Ha, Thanh T.
Group Art Unit:	2194
Conf. No.:	8094

Certificate of Mailing Under 37 C.F.R. §1.8

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: **MAIL STOP APPEAL BRIEF PATENTS**, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on:

Date: February 8, 2006 By: Farah Z. Frasco
(Typed or printed name of person mailing Document, whose signature appears below)

Signature: _____

MAIL STOP APPEAL BRIEF PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

(i) Real party in interest.

EMC Corporation, a corporation organized and existing under the laws of the Commonwealth of Massachusetts and having a usual place of business at 35 Parkwood Drive, Hopkinton, Massachusetts 01748.

(ii) Related appeals and interferences.

None

(iii) Status of Claims

Claims 1-35 stand rejected under 35 U.S.C. 102(e) as being unpatentable over Young (U.S. Patent 6,782,531).

(iv) Status of Amendments.

No amendment has been filed subsequent to mailing of the final rejection on August 22, 2005.

(v) Summary of Claimed Subject Matter

The subject application includes 4 independent claims, namely, claims 1, 15, 29, and 30.

The subject matter associated with claims 1 and 15 is described in the specification at page 13, line 12 to page 14, line 24 as well as elsewhere throughout the application. This passage of text refers to corresponding FIGS. 1 and 2 summarizes the claimed invention as follows:

The present invention as in claim 1 provides techniques and mechanisms that enable a plug-in manager executing on a respective computer system to manage services associated with a plurality of plug-in modules. By way of example, one embodiment of the system of the invention is described via the following steps.

In step 202 of FIG. 2 of the subject application, the plug-in manager 150 obtains the identities of a plurality of plug-in modules 160 based upon the plug-in services identified in the list of services 170. As an example, the names of the particular plug-in services may correspond to the identity of plug-in module definitions 162 defined within the plug-in database 155.

In step 203, the plug-in manager retrieves a dependency list indicating respective plug-in services provided by, and required by, each plug-in module 160 identified in the identities of the plurality of plug-in modules (obtained in step 202). To perform step 203, the plug-in manager process 150 can access the plug-in module

definitions 162 (e.g., Java classes) to instantiate respective plug-in modules 160 corresponding to the services identified in the request 170. The plug-in modules 160 are instantiated into an enabled state in which each plug-in module 160 is resident in memory within the server computer system 120 but is not yet providing its respective service 147. Once each plug-in module 160 is operating in the enabled state, the plug-in manager 150 can provide a dependency query 171 to each plug-in module 160 to determine what services that plug-in module 160 provides and to also determine what services that plug-in module 160 may require to properly operate. In response to such a query, each plug-in module 160 can provide a dependency response 172 back to the plug-in manager process 150 indicating services provided and services required. The collective set of services provided by and required for each respective plug-in module 160 comprises a dependency list compiled, retrieved or otherwise developed by the plug-in manager process 150.

Next, in step 204, the plug-in manager process 150 calculates a plug-in initiation order based upon the dependency list that indicates the respective plug-in services 147 provided by and required by each plug-in module 160. In other words, in step 204, the plug-in manager process 150 figures out an order in which plug-in modules 160 must be transitioned from the enabled state into a started state in which they can provide their respective services, such that plug-in modules that depend upon or require the services 147 of other plug-in modules are started after the plug-in modules 160 providing those required services.

Next, in step 205, the plug-in manager process 150 initiates service operation of plug-in modules 160 according to the plug-in initiation order calculated according to the processing performed in step 204. The plug-in manager process 150 initiates the service operation 147 of the plug-in modules 160 in step 205 such that if a first plug-in module, such as plug-in module 160-1, provides a service (e.g., 147-1, not specifically labeled) that is required by a second plug-in module 160-2, the first plug-in module 160-1 is initiated such that the service 147-1 is available to the second plug-in module 160-2 when required by the second plug-in module 160-2. Generally, in such a case, the first plug-in module 161 is initiated into the started or "service operation" state before the second plug-in module 160-2. The plug-in

manager 150 can initiate service operation in a plug-in module 160 by sending a "dependency available" message 173 to that plug-in module 160. The dependency available message 173 indicates the services that are now available for use by that particular plug-in module 160. These may be, for example, the services that that particular plug-in module 160 requires to properly operate.

Claim 29 includes the same limitations as claim 1. Thus, the above-cited passage for claim 1 describes the first four elements of claim 29. Additional support for the last element of claim 29 can be found at page 22, line 21 to page 23, line 3 of the specification, which reads as follows:

In step 308, the plug-in initializer 152 queries a dependency interface associated with the plug-in module 160 using a dependency query 171 to obtain a dependency response 173 from the plug-in module 160. In the example of plug-in module architecture illustrated in Figure 4, the dependency interface is the service interfaces 161 for each respective plug-in service that the plug-in module 160 provides, as well as the list of required services 162 that the plug-in module 160 requires to properly operate. In other words, in step 308, the plug-in initializer 152 accesses the service interfaces 161 and required services 162 within the plug-in module 160 to determine which services are provided by, and required by, the plug-in module 160.

Next, in step 309, the plug-in module receives a dependency response 173 from the plug-in module 160. The dependency response 173 indicates respective plug-in services provided by, and required by, the plug-in module 160.

Claim 30 includes the same limitations as claim 1. Thus, the above-cited passage for claim 1 describes the first four elements of claim 30. Additional support for the last element of claim 30 can be found at page 17, line 18 to page 18, line 5 of the specification, which reads as follows:

In addition, in this example plug-in module 160, a wait state operation 165 is included which represents processing and logic instructions required to allow this

plug-in module 160 to determine if other plug-in modules upon which it depends for operation are available for service access. In other words, a wait state operation 165 represents the processing functionality to allow this plug-in module 161 to signal another plug-in module that it is available for service processing or that it is waiting for that other plug-in module to provide a particular required service. As will be explained further, the wait state operation 165 is the useful in situations where the plug-in manager process 150 detects an interdependency between two plug-in modules 160. If the plug-in manager process 150 determines that two plug-in modules depend upon each other for services 147, the plug-in manager process 150 can cause each plug-in module to enter the started or service providing state and can then invoke the wait state operation 165 within one or more of these inter-dependent plug-in modules 160 to allow those plug-in modules to intercommunicate with each other to indicate when each is available to begin providing a service upon which the other plug-in modules depend. In other words, the wait-state operation interface 165 allows the plug-in manage to pass-off the duties of coordinating a smooth (i.e., non-deadlocked) service start-up procedure between two or more plug-in modules that depend upon each other to those plug-in modules 160.

In addition to the cited passages above, additional support for claims 2-34 can be found in the summary of invention, which is directed for use in environments including plug-in modules. In particular, embodiments of the invention provide a plug-in manager application and process that operates in a computer system to manage services associated with a plurality of plug-in modules. The plug-in manager process can operate on behalf of software applications requiring access to plug-in services by receiving, detecting or otherwise determining a list of plug-in services that the software application requires. The plug-in manager can then query a set of plug-in modules identified in the list in order to determine service dependencies between the plug-in modules. For example, if a software application indicates that three plug-in services are required during operation of that software application, the plug-in manager process can operate the plug-in modules in an initiation or enabled mode and can query those plug-in modules for a list of services that they provide and for a list services that they will require when operating in a service operation mode. Using this information, if necessary the plug-in manager

can also query other plug-in modules which were not identified by the software application, but which supply or provide the services that are required by the plug-in modules that the software application did identify. This process can continue until the plug-in manager discovers and queries all plug-in modules.

Once the plug-in manager obtains the dependencies of all plug-in modules, the plug-in manager can calculate a plug-in initiation sequence or order based upon the dependencies identified between the plug-in modules. This may be done, for example, by analyzing the plug-in module dependencies to produce a dependency tree that contains nodes that represent plug-in modules and whose hierarchy represents a plug-in module initiation order. For example, the lowest or leaf notes in the plug-in module dependency tree identify services of plug-in modules that should be initiated or started first whereas upper-level nodes in the tree identify plug-in modules that provide services that should be initiated after their child nodes in the dependency tree.

Using the calculated plug-in initiation order, the plug-in manager can initiate service operation of the plug-in modules according to the plug-in initiation order. As a result, if a first plug-in module provides a service required by a second plug-in module, the first plug-in module will be initiated such that the service provided by the first plug-in module is available to the second plug-in module when required by the second plug-in module. That can result, for example, in plug-in modules that provide services that other plug-in modules rely upon being started prior to the plug-in modules that require those services. In this manner, embodiments of the invention help to insure that plug-in modules are initiated in a proper sequence so that all services required by plug-in modules will be available when the plug-in modules requiring those services are initiated.

In some instances, if two plug-in modules rely on the services of each other, the initiation order may be to start the services of the plug-in modules concurrently or, one before the other. In such a situation, one plug-in module can include a wait state operation that the plug-in manager can operate to cause the plug-in module to wait to provide its service until the other plug-in module begins providing service.

Alternatively, the wait state operation can cause one plug-in module to signal to the other plug-in module (the other one that is dependent upon the one providing the signal) to indicate that a service is not available. This allows two plug-in modules that depend upon each other to signal to each other to allow one to start followed by the other in an order which the plug-in modules can determine to avoid a deadlock in the plug-in module service initiation order.

In addition, certain embodiments of the invention operate using Java or other object-oriented interfaces within the computer system environment for communication between the plug-in manager and the plug-in modules. Using such interfaces avoids having to incorporate different proprietary code for multiple interfaces to different proprietary plug-in modules.

Generally, one method embodiment of the invention operates to manage plug-in services by obtaining identities of a plurality of plug-in modules and retrieving a dependency list indicating respective plug-in services provided by, and required by, each plug-in module identified in the identities of a plurality of plug-in modules. As noted above, a plug-in manager of performing this method is a thus able to determine what services are provided and required by all plug-in modules. Thereafter, the method calculates a plug-in initiation order based upon the dependency list indicating respective plug-in services provided by, and required by, each plug-in module. The method then initiates service operation of plug-in modules according to the plug-in initiation order, such that if a first plug-in module provides a service required by a second plug-in module, the first plug-in module is initiated such that the service provided by the first plug-in module is available to the second plug-in module when required by the second plug-in module. Additional support can be found elsewhere throughout the specification.

Additional support for claims 3 and 17 can be found in FIGS. 2, 5-7 as well as at page 12 line 19 to page 15 line 15 and page 18 line 6 to page 24 line 9 as well as elsewhere throughout the specification.

Additional support for claims 5 and 19 can be found in FIGS. 2, 5-7 as well as at page 12 line 19 to page 15 line 15 and page 18 line 6 to page 24 line 9 as well as elsewhere throughout the specification.

Additional support for claims 7 and 21can be found in FIGS. 6-7 as well as at page 19 line 22 to page 24 line 9 as well as elsewhere throughout the specification.

Additional support for claims 10 and 24 can be found in FIGS. 3, 6-7 as well as at page 15 line 15 to page 16 line 22 as well as at page 19 line 22 to page 24 line 9 as well as elsewhere throughout the specification.

Additional support for claims 13 and 27 can be found in FIG. 4 and 8 as well as at page 16 lines 22 t page 18 line 6 as well as at page 24 line to page 28 line 5 and elsewhere throughout the specification.

Additional support for claims 31-32 can be found at page 5 lines 1-16 and elsewhere throughout the specification. Support for claims 33-34 can be found at page 6 lines 6-16 and elsewhere throughout the specification. Support for claims 35 can be found at page 10 lines 16-22 and elsewhere throughout the specification.

(vi) Grounds of Rejection to be reviewed on Appeal.

The Examiner rejects claims 1-35 under 35 U.S.C. §102(e) as being anticipated by Young (U.S. Patent 6,782,531).

(vii) Argument

Note that the Applicants attempt to set forth the Examiner's argument to the best of their ability. Unfortunately, the arguments made in the Final office action did not identify to which claim they pertained. It is therefore unclear as to which points belong to which claims. Also, the Applicants respectfully submit that the Examiner improperly restates the Applicants arguments regarding patentability.

Rejection of Claim 1

The Examiner rejects claim 1 based on the following arguments in the Final Office action dated August 22, 2005:

> As to claim 1, Young teaches the invention as claimed including a computer system, a method for managing services associated with a plurality of plug-in modules [abstract, lines 1-3], the method comprising:
>
> obtaining identities of a plurality of plug-in modules [col. 6, lines 19-21];
>
> based on queries to the plurality of plug-in modules, retrieving a dependency list indicating respective plug-in services provided by, and required by, each plug-in module identified in the identities of a plurality of plug-in modules [col. 2, lines 57-61];
>
> calculating a plug-in initiation order based upon the dependency list indicating respective plug-in services provided by, and required by, each plug-in module [col. 2, line 59-65]; and
>
> initiating service operation of plug-in modules according to the plug-in initiation order, such that if a first plug-in module provides a service required by a second plug-in module, the first plug-in module is initiated such that the service provided by the first plug-in module is available to the second plug-in module when required by the second plug-in module [col. 8, lines36-38].

In response to the arguments made by the Applicant regarding patentability, the Examiner further argues:

> As to point (a), Young clearly teaches retrieving a dependency list based on queries of the plug-in modules as described in the following lines:
>
> "a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in" [col. 13, lines 60-61].

Applicants respectfully submit that the cited technique in Young is not equivalent to the limitation in the claimed invention. More specifically, the claimed invention is directed (at least in part) towards retrieving a dependency list based on queries to a plurality of plug-in modules. The portion of Young cited by the Examiner and set forth to reject the claimed invention recites that a mere "database query" emanates from a plug-in module to a database. These operations (e.g., the claimed technique versus the cited prior art) occur in

opposite directions with respect to each other and therefore are not equivalent. Also, the type of information associated with a respective data transfer (e.g., with respect to the claimed invention versus cited prior art) are also different. For example, the data transfer in the claimed invention includes retrieval of dependency information from the plug-in modules, while the communication by the plug-in module in Young is a "database query" and does not include dependency information.

The Examiner originally rejected this claim element based on [col. 2, line 57-61] of Young. This cited passage only indicates that Young includes an "order determining mechanism" for determining an operational sequence of when plug-in modules. Applicant respectfully submits that Young at column 8, lines 26-30 discloses details of the "order determining mechanism, which is not defined as a plug-in module. For example, Young at column 8, lines 26-30 states: "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as the operations of the plug-ins 1-8 of the corresponding process space 404 and their processing inter-dependencies." There is no indication that the configuration manager 150 in Young communicates with the plug-in to retrieve dependency information.

In rebuttal to the argument (e.g., point (a)) made by the Examiner in the Final Office Action, Applicant respectfully traverses the rejection based on grounds that the cited prior art does not teach all of the claim limitations and that the Examiner improperly restates Applicants' arguments in the last office action response. The Examiner contends that column 13, lines 60-61 in Young (e.g., "a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in" emphasis added by the Examiner) reads on the claim limitation of "based on queries to the plurality of plug-in modules, retrieving a dependency list indicating respective plug-in services provided by, and required by, each plug-in module identified in the identities of a plurality of plug-in modules" as in claim 1.

First, the claimed technique recites retrieving a dependency list based on queries sent to the plurality of plug-in modules. The passage cited in Young merely indicates that a plug-in must wait for results of a database query by another plug-in module. Thus, claim 1 recites a technique that operates in an opposite manner as the cited subject matter used to reject the

claimed invention. For example, the claimed invention indicates that the queries are sent to the plug-in modules. The cited reference set forth to reject that recites that a "database query" emanates from a plug-in module to a database. The database in Young is not a plug-in module. See FIG. 6B in Young illustrating that the dependency data 654 is stored in execution management framework 652. Note that plug-in modules 662, 664, and 666 reside at a remote location with respect to the execution management framework 652 and there is no indication that the plug-in modules provide dependency information to the execution management framework 652 or any other entity in Young. Thus, based on these grounds, the concepts are not equivalent and the rejection under 35 U.S.C. §102 is improper.

Furthermore, note that the queries to the plug-in modules (according to the claimed invention) prompt retrieval of a dependency list, which is defined in the claim as a different kind of information as that recited in Young. For example, according to claim 1, the retrieved dependency list indicates respective plug-in services that are provided by and required by each of the plurality of identified plug-in modules. The "database query by another plug-in module" in Young is not made for purposes of retrieving dependency requirements of or services provided by plug-in modules. Thus, for this additional reason, Young does not teach every claim limitation.

This claimed technique enables the plug-in modules themselves to provide dependency information of what other plug-in modules to execute rather than relying on efforts of the execution management framework as in Young to track such information for all plug-ins via a configuration file. Also, the plug-in modules provide an indication of what services are supported by respective plug-in modules based on the queries to the plug-in modules. This is also not supported by Young. Thus, Applicants respectfully request the withdrawal of the respective rejection of claim 1 under 102(e).

For the reasons stated above, Applicants submit that claim 1 includes limitations not found in any of the cited references and therefore is patentably distinct and advantageous over the cited prior art. Applicants respectfully request the withdrawal of the rejection of claim 1 under 35 U.S.C. §102(e). Accordingly, allowance of claim 1 is respectfully requested.

By virtue of dependency, respective dependent claims 2-14 share also in condition for allowance. Specific opposition to the rejection of certain dependent claims follows below.

Rejection of Claim 15

The Examiner rejects claim 15 based on the same reasons discussed above for claim 1.

Applicants respectfully submit that corresponding apparatus claim 15 recites that the corresponding steps (e.g., steps similar to those recited in claim 1) are executed in a plug-in manager application. Applicants again respectfully submit that there is no indication in the cited passages that a plug-in manager retrieves the dependency and service information based on queries to each of multiple plug-in modules for the same reasons discussed above for claim 1.

Rejections of Claims 3 and 17

The Examiner rejects claim 3 and 17 based on the following arguments in the Final Office action dated August 22, 2005:

> As to claim 3, Young teaches wherein the step of retrieving a dependency list indicating respective plug-in services provided by, and required by, each plug-in module comprises the steps of:
>
> for each plug-in module identified in the identities of a plurality of plug-in modules, performing the steps of:
>
> instantiating the plug-in module based upon a plug-in module definition associated with the identity of the plug-in module[col. 13, lines 53-54] ;
>
> receiving a dependency response from the plug-in module, the dependency response indicating respective plug-in services provided by, and required by, the plug-in module [col. 13, line 50-53] ; and
>
> storing identities of the respective plug-in services provided by, and required by, the plug-in module as identified in the dependency response in the dependency list [col. 13, line 50].

Applicants submit that claim 3 includes further patentable limitations not disclosed by Young. For example, claim 3 recites "receiving a dependency response from the plug-in module..." The Office Action cites column 13, lines 50-54 to reject this portion of claim 3 of the subject application. The cited language in Young reads as follows:

"The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed." (emphasis added)

Applicants respectfully submit that this passage provides no literal support or suggestion that any plug-in module as in Young provides "a dependency response from a plug-in module…indicating respective plug-in services provided by, and required by, the plug-in module" as in the claimed invention. That is, the Examiner has not cited language in Young indicating that the "configuration file" resides in any respective plug-in module. FIG. 6b of Young indicates that the dependency data 654 resides in the execution management framework, which is not located in a respective plug-in module. Note also that there no indication in the cited Young reference that any plug-in module itself provides an indication of services supported by the plug-in module.

In a similar vein as discussed for claim 1, the technique in claim 3 enables the plug-in modules themselves to provide dependency information of what other plug-in modules to execute rather than relying on efforts of the execution management framework to track such information for all plug-ins via maintaining a configuration file. Thus, Applicants respectfully request that respective rejections of claim 3 and 17 under 102(e) be withdrawn.

Rejections of Claims 5 and 19

The Examiner rejects claim 5 and 19 based on the following arguments in the Final Office action dated August 22, 2005:

As to claim 5, Young teaches wherein the step of instantiating the plug-in module comprises the step of:

querying a dependency interface associated with the plug-in module with a dependency query to obtain the dependency response from the plug-in module [col. 13, line 61].

Applicants submit that claims 5 and 19 include further limitations not disclosed by Young. For example, claims 5 and 19 recites "querying a dependency interface ... to obtain the dependency response from the plug-in module." As recited in claim 3, the dependency response indicates respective plug-in services provided by and required by the respective plug-in module. The Office Action cites column 13, line 61 to reject this claim limitation. This cited passage merely indicates that the pipeline infrastructure can execute another plug-in module if a plug-in module will take a long time to execute because it must wait on the results of a database query by another plug-in module. As discussed above, there is no indication that the database query in Young is generated for the purpose of obtaining dependency information from the plug-in module as in the claimed invention. Nor is the database query executed in Young generated for a purpose of learning which services are supported by a respective plug-in module. The cited passage therefore does not teach or suggest the claimed invention. Thus, Applicants respectfully request the withdrawal of the respective rejection of claims 5 and 19 under 102(e) should be withdrawn.

Rejections of Claims 7 and 21

The Examiner rejects claim 7 and 21 based on the following arguments in the Final Office action dated August 22, 2005:

As to claim 7, Young teaches wherein the step of arranging placement of each plug-in module identified in the dependency list within the plug-in initiation order comprises the steps of:

analyzing the dependency list indicating respective plug-in services provided by, and required by, each plug-in module to determine which plug-ins provide services relied upon by other plug-in modules [col. 13, line 31-44] ; and

creating, as the plug-in initiation order, at least one plug-in module dependency tree based on the step of analyzing, the at least one plug-in module dependency tree containing a hierarchical arrangement of nodes associated with respective plug-in

modules, the hierarchical arrangement indicating the plug-in initiation order of the plug-in modules respectively associated with the nodes in the dependency tree [col. 13, line 44-49].

The cited passage in Young to reject the last element of claims 7 and 21 reads as follows: "The dependencies noted in the graph can be sorted and listed in the stage configuration file, as described below. Alternatively, the graph 600 of the plug-ins can be represented, e.g., by a suitable data structure, array or table as part of the configuration file for the particular stage in configuration storage 508 (FIG. 5)."

Applicants respectfully submit that claims 7 and 21 that the graph, data structure, array or table recited in Young is not equivalent to the dependency tree containing a hierarchical arrangement of corresponding nodes as recited in the claimed invention. Accordingly, the rejection under 102 is improper.

Applicants respectfully submit that Young at these cited passages also does not suggest the claimed invention and a 103 rejection would be improper. For example, the claimed invention includes a plug-in dependency tree in which a hierarchical arrangement of nodes indicates an initiation order of the plug-in modules. Based on such a topology, an initiation order can be quickly identified based on the arrangement of hierarchical nodes associated with respective plug-in modules in the dependency tree. The cited passage in Young (e.g., "a suitable data structure") states a mere truism that dependency data can be stored for later retrieval. Since the statement in Young is only a truism, there is no suggestion that a position of nodes in a respective data structure indicates an initiation order. Accordingly, Applicants respectfully submit that an obviousness rejection would be also improper for these claims.

Applicants respectfully request the withdrawal of the respective rejections of claim 7 and 21 under 102(e) and allowance of these claims.

Rejections of Claims 9 and 23

The Examiner rejects claim 9 and 23 based on the following arguments in the Final Office action dated August 22, 2005:

As to claim 9, Young teaches wherein the step of initiating service operation of plug-in modules includes:

forwarding, via a dependency available interface associated with a respective plug-in module, a list of initiated plug-in services of other plug-in modules that are currently available for use by the respective plug-in module [col. 8, lines 31-38].

Claim 9 recites "forwarding, via a dependency available interface associated with a respective plug-in module, a list of initiated plug-in services of other plug-in modules that are currently available for use by the respective plug-in module." To reject this claim, the Examiner cites Young at column 8, lines 31-38 which states:

"The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins." (emphasis added)

Applicants traverse the rejection on grounds that the cited passage in Young does not recite the limitations as purported by the Examiner. For example, the claim recites forwarding a list of services to a respective plug-in module indicating which services are available from other plug-in modules. Consequently, the respective plug-in modules according to the claimed invention know which services are available from the other plug-in modules. The execution management framework 425 in Young is not a plug-in module. Moreover, the execution management framework 425 does not communicate such information to any of the plug-in modules. Thus, the rejection under 102 is improper.

Rejections of Claims 10 and 24

The Examiner rejects claims 10 and 24 based on the following arguments in the Final Office action dated August 22, 2005:

As to claim 10, Young teaches wherein the step of initiating service operation of plug-in modules according to the plug-in initiation order comprises performing, for each respective plug-in module in the plug-in initiation order, the steps of:

determining, from a published list of services available by initiated plug-in modules, an identity of each initiated plug-in service required by the respective plug-in module [col. 7, lines 59-60] ;

forwarding to the respective plug-in module, via a dependency available interface associated with the respective plug-in module, the identity of each initiated plug-in service required by the respective plug-in module [col. 8, lines 31-32] ;

receiving a list of services initiated by the respective plug-in module [col. 8, lines 15-16]; and

adding the list of services provided by the respective plug-in module to the published list of services [col. 8, lines 17-18].

The Examiner further argues:

As to point (c), Young clearly teaching "forwarding ("passing") an identity of each initiated plug-in service". The plug-in number (1-8) in Young is used as the identity of each initiated plug-in service. [col. 8, lines 30-38].

Regarding point (C) made by the Examiner in the final office action, Applicants submit that claim 10 and, perhaps more importantly, claim 24 include limitations not disclosed by Young. For example, both claims 10 and 24 recite "forwarding to the respective plug-in module, via a dependency available interface associated with the respective plug-in module, the identity of each initiated plug-in service required by the respective plug-in module." Additionally, claim 24 specifically recites that a plug-in manager application supports operations of "forwarding to the respective plug-in module, via a dependency available interface associated with the respective plug-in module, the identity of each initiated plug-in service required by the respective plug-in module." The above-cited passage associated with Examiner's point (c) merely indicates that the plug-in modules have been named (e.g., numbered). This is not equivalent to the claim limitations in the claimed invention.

The Office Action cites column 8, lines 31-32 to reject this portion of claim 10 of the subject application. The cited language in Young reads as follows:

> "The stage configuration module passes the plug-in configuration information to an execution management framework 425."

Applicants respectfully submit that this passage provides no literal support associated with forwarding an identity of each initiated plug-in service required by the respective plug-in module. FIG. 4 more particularly illustrates that the stage configuration module 416 (see column 8, lines 25-27) passes plug-in configuration information to the execution management framework 425.

The Examiner further cites Young at column 8, lines 30-38 which reads as follows:

> "The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins."

First, the execution management framework 425 in Young is not a plug-in module. Claim 10 recites forwarding information from a plug-in module manager to a respective plug-in module. Thus, Young does not disclose this aspect of the claimed invention.

Second, contrary to the Examiner's assessment, the execution management framework 425 does not forward an identity of plug-in modules to a respective plug-in module. Instead, as recited in Young, the execution management framework 425 merely determines which of multiple plug-in modules can be processed in parallel and which must be processed in a particular sequence. The execution management framework 425 does not forward any identity information to the plug-in modules. As its name suggests, the execution management framework 425 initiates execution of the plug-in modules nos. 1-8 (from a remote location with respect to any plug in modules in a sequence or in parallel by sending the plug-in modules a respective execution command and counter value that is used to determine when to execute the respective plug-in module over time. There is no indication

that the plug-in modules are aware of the identity of which other plug-in modules are required by the remote server plug-in module.

The claimed technique as recited in claims 10 and 24 enable the plug-in modules to manage themselves based on knowing which plug-in module from which they depend. The plug-in modules in Young only understand at what point in time they are to execute and not from which other plug-in modules they depend. Thus, Applicants respectfully request that Examiner allow claims 10 and 24 over Young.

Rejections of Claims 13 and 27

The Examiner rejects claims 13 and 27 based on the following arguments in the Final Office action dated August 22, 2005:

As to claim 10, Young teaches wherein the first plug-in module is initiated via the step of initiating operation of plug-in modules after initiation of the second plug-in module, and wherein the second plug-in module includes a wait-state operation causing the second plug-in module to wait to provide the service offered by the second plug-in module until initiation of the first plug-in module [col. 13, lines 28-30].

The Examiner further argues:

As to point (d), Applicant invention states that:

"the second plug-in module requires a service provided by the first plug-in module" (as claimed in claim 11 and also in Applicant amendment page 21, last paragraph "e.g. the second plug-in module that utilizes services offered by the first plug-in")".

"plug-in N depends on plug-in M" (col. 13, lines 25-26). Therefore, the plug-in M corresponds to Applicant's second plug-in and the plug-in M corresponds to Applicant's first plug-in.

The claimed language does not recite that the "depending" plug-in is executed before executing the plug-in on which it depends. Rather, the claimed language states that the "depending" (second) plug-in module wait to provide the service until initiation of the first plug-in module. Therefore, the first plug-in (M) is executed first before executing the second plug-in (N). And, that is exactly Young teaching:

"plug-in M to be executed before it starts execution of plug-in N" (col. 13, lines 29-30).

In addition, wait-state is not considered an execution state. According to Microsoft Computer Dictionary (Fifth Edition):

"execute-To perform an instruction".

Wait-state does not required to perform any instruction. Therefore, the second plug-in does not executed first.

Regarding point (d) made by the Examiner, Applicants again submit that claim 13 includes further limitations not found in Young. For example, the claimed invention of claim 13 recites: "wherein the first plug-in module is initiated via the step of initiating operation of plug-in modules after initiation of the second plug-in module, and wherein the second plug-in module includes a wait-state operation causing the second plug-in module to wait to provide the service offered by the second plug-in module until initiation of the first plug-in module."

The passages cited by the Examiner to reject claim 13 include Young at column 13, lines 28-30 which read as follows:

"As noted above, plug-ins can be implemented as modular pieces of code that are executed during run-time for performing a defined task, such as a sub-computation on session data. Usually, the plug-ins need to be executed in a certain, specified order to effectuate the desired, overall computation performed by the stage that contains them. A complexity is introduced in specifying that order because plug-ins can be dependent on other plug-ins. Generally speaking, given two plug-ins M and N, if plug-in M computes the value x (as a final or intermediary result) and plug-in N requires the value x to perform its computation, then plug-in N depends on plug-in M. Plug-ins can be dependent on zero, one, two, or more, other plug-ins. In the above notation, because of the noted dependency between M and N, the stage infrastructure will wait for plug-in M to be executed before it starts execution of plug-in N. Plug-ins with zero dependencies can be executed immediately or at any other time, without regard to prior execution of other plug-ins." (Emphasis added)

Based on the cited passage, Applicants submit that the claimed invention operates in a different manner than as mentioned in Young. For example, according to the cited passage, Young indicates that plug-in N depends from plug-in M and that the "stage infrastructure" will wait for plug-in M to be executed before it starts execution of plug-in M. This means that the stage infrastructure executes plug-in M first and "depending" plug-in N later in time. Applicants are unable to the follow the logic used to reject the claimed invention because the Examiner's makes an argument and agrees that "the first plug-in (M) is executed first before executing the second plug-in (N)." Applicants again submit that Young discloses a technique opposite to that recited in the claimed invention.

First, the claimed invention recites that the "depending" plug-in module (i.e., the second plug-in module as recited in claim 13) is initiated first before the non-depending plug-in module (i.e., the first plug-in module as recited in claim 13). This is opposite to the configuration in Young where the "depending" plug-in N is executed after plug-in M.

Second, the claimed invention recites that the second plug-in module includes a wait operation causing the second plug-in module to wait to provide the service offered by the second plug-in module until initiation of the first plug-in module. Young does not recite this technique of waiting because there would be no need to wait; the "depending" plug-in module in Young is executed after the plug-in module on which it depends after data is available. Thus, there would be no need to wait in Young as recited in the claimed invention.

Based on the recited technique in claim 13, there is no longer a need to initiate execution of the plug-in modules in any particular order, even though a dependency exists. Young makes no mention of initiating execution of plug-ins in the order as described in the claimed invention. Nor does Young discuss a wait type of operation associated with a plug-in to receive services offered by another plug-in. Providing a wait-state operation in Young would serve no useful purpose because it would not be necessary.

For the reasons stated above, Applicants submit that claim 13 and claim 27 each includes limitations not found in any of the cited references and therefore is patentably distinct and advantageous over the cited prior art. Applicants respectfully request the

withdrawal of the rejection of claim 13 under 35 U.S.C. §102(e) or request that the Examiner more particularly point out passages in Young that teach the claimed invention. Accordingly, allowance of claim 13 and claim 17 is respectfully requested.

Rejection of Claim 29

The Examiner rejects claim 29 based on the following arguments in the Final Office action dated August 22, 2005:

> As to claim 29, this is a computer program product claim that corresponds to method claim 1. Therefore, it is rejected for the same reason as claim 1 above.

The Examiner provides no particular argument regarding the rejection of claim 29 other than that provided for claim 1. This is an incorrect assessment. Claim 29 includes limitations not found in claim 1.

For example, Applicant would like to point out that claim 29 (in addition to the elements in claim 1) recites "querying a dependency interface associated with the plug-in module with a dependency query to obtain a dependency response from the plug-in module, the dependency response indicating respective plug-in services provided by the plug-in module." In short, Young does not disclose querying a plug-in module to learn of plug-in services provided by the plug-in module as discussed above. Applicants respectfully request allowance of claim 29.

Rejection of Claim 30

The Examiner rejects claim 30 based on the following arguments in the Final Office action dated August 22, 2005:

> As to claim 30, this is a computer system claim that corresponds to method claim 1. Therefore, it is rejected for the same reason as claim 1 above.

The Examiner provides no particular argument regarding the rejection of claim 30 other than that provided for claim 1. This is an incorrect assessment. Claim 30 includes limitations not found in claim 1.

For example, Applicants would like to point out that claim 30 (in addition to the elements in claim 1) recites "wherein the first plug-in module is initiated via the step of initiating operation of plug-in modules after initiation of the second plug-in module, and wherein the second plug-in module includes a wait-state operation causing the second plug-in module to wait to provide the service offered by the second plug-in module until initiation of the first plug-in module." In short, Applicant respectfully submits that Young does not disclose this limitation for the reasons set forth above for claim 13. Applicants respectfully request allowance of claim 30.

Rejection of Claim 31

The Examiner rejects claim 31 based on the following arguments in the Final Office action dated August 22, 2005:

> As to claim 31, A computer program product as in claim 29, wherein the processor further performs operations of:
>
> determining a list of plug-in services required by a software application [col. 14, lines 47-50]; and
>
> querying a set of plug-in modules to identify services provided by the set of plug-in modules [col. 13, lines 50-63].

Applicants respectfully submit that claim 31 includes limitations not found in Young. For example, claim 31 recites "querying a set of plug-in modules to identify services provided by the set of plug-in modules." In short, contrary to the Examiner's assessment of the cited passage, as discussed above, Young does not disclose that a management entity or any other entity queries any plug-in modules to learn of services performed by the plug-in modules.

Rejection of Claim 32

The Examiner rejects claim 32 based on the following arguments in the Final Office action dated August 22, 2005:

> As to claim 32, A computer program product as in claim 31, wherein the processor further performs operations of:

in response to querying the set of plug-in modules, identifying plug-in modules not identified by the software application as being necessary but which are identified by the set of plug-in modules as being necessary to carry out execution of an operation on behalf of the software application [col. 14, lines 43-64].

Applicant respectfully submits that claim 32 includes limitations not found in Young. For example, claim 32 recites "in response to querying the set of plug-in modules, identifying plug-in modules not identified by the software application as being necessary but which are identified by the set of plug-in modules as being necessary to carry out execution of an operation on behalf of the software." In short, contrary to the Examiner's assessment of the cited passage, Young does not disclose that a set of plug-in modules identifies which plug-in modules are necessary to be carried out in order to carry out execution of an operation on behalf of the software application.

Rejection of Claim 33

The Examiner rejects claim 33 based on the following arguments in the Final Office action dated August 22, 2005:

As to claim 33, A computer program product as in claim 32, wherein the processor further performs operations of:

initiating service operation of plug-in modules on the processor according to an order other than the plug-in initiation order, such that if a third plug-in module provides a service required by a fourth plug-in module, the third plug-in module being initiated after initiation of the fourth plug-in module, the third plug-in module initiating a wait state operation causing the third plug-in module to wait to provide the service offered by the third plug-in module until instantiation of the fourth plug-in module [col. 13, lines 14-64].

For example, claim 33 recites that "the third plug-in module initiating a wait state operation causing the third plug-in module to wait to provide the service offered by the third plug-in module until instantiation of the fourth plug-in module." Contrary to the Examiner's assessment of the cited passage, Young does not disclose that any plug-in modules initiate execution of a wait state operation, especially one in which a respective plug-in module waits for instantiation of another plug-in module to provide a service. Nor does Young recite the other limitations as in claim 33.

Rejection of Claim 34

The Examiner rejects claim 34 based on the following arguments in the Final Office action dated August 22, 2005:

> As to claim 34, A computer program product as in claim 32, wherein the processor initiates execution of the first plug-in module before execution of the second plug-in module, the first plug-in module initiating a wait state operation resulting in signaling to the second plug-in module, the signaling indicating that a respective service of the first plug-in module is not yet available to the second plug-in module [col. 13, lines 14-64].

For example, claim 34 recites "wherein the processor initiates execution of the first plug-in module before execution of the second plug-in module, the first plug-in module initiating a wait state operation resulting in signaling to the second plug-in module, the signaling indicating that a respective service of the first plug-in module is not yet available to the second plug-in module." Contrary to the Examiner's assessment of the cited passage, and for the reasons discussed above, Young does not disclose that any plug-in modules themselves initiate execution of a wait state operation. Instead, Young discloses that a management framework associates count values for purposes of determining when to execute different plug-in modules.

(viii) Claims Appendix

Attached below.

(ix) Evidence Appendix

None

(x) Related Proceedings Appendix

None

The Assistant Commissioner is hereby authorized to charge payment of any additional fees associated with this communication or credit any overpayment to Deposit Account No. 50-3735.

Respectfully submitted,

Date: February 8, 2006

Paul P. Kriz, Esq.
Attorney for Applicant(s)
Registration No.: 45,752
Chapin Intellectual Property Law, LLC
Westborough Office Park
1700 West Park Drive
Westborough, Massachusetts 01581
Telephone: (508) 616-9660
Facsimile: (508) 616-9661

Appendix of Pending Claims at Mailing of Final Office Action

1. (Previously Presented) In a computer system, a method for managing services associated with a plurality of plug-in modules, the method comprising the steps of:

obtaining identities of a plurality of plug-in modules;

based on queries to the plurality of plug-in modules, retrieving a dependency list indicating respective plug-in services provided by, and required by, each plug-in module identified in the identities of a plurality of plug-in modules;

calculating a plug-in initiation order based upon the dependency list indicating respective plug-in services provided by, and required by, each plug-in module; and

initiating service operation of plug-in modules according to the plug-in initiation order, such that if a first plug-in module provides a service required by a second plug-in module, the first plug-in module is initiated such that the service provided by the first plug-in module is available to the second plug-in module when required by the second plug-in module.

2. (Previously Presented) The method of claim 1 wherein the step of obtaining identities of a plurality of plug-in modules includes the steps of:

receiving a list of services to be started within the computer system ;

determining, for each service in the list of services, a respective plug-in module definition that can provide that service; and

placing the identity of each plug-in module definition determined in the step of determining into the identities of the plurality of plug-in modules.

3. (Original) The method of claim 1 wherein the step of retrieving a dependency list indicating respective plug-in services provided by, and required by, each plug-in module comprises the steps of:

for each plug-in module identified in the identities of a plurality of plug-in modules, performing the steps of:

instantiating the plug-in module based upon a plug-in module definition associated with the identity of the plug-in module;

receiving a dependency response from the plug-in module, the dependency response indicating respective plug-in services provided by, and required by, the plug-in module; and

storing identities of the respective plug-in services provided by, and required by, the plug-in module as identified in the dependency response in the dependency list.

4. (Original) The method of claim 3 wherein the step of instantiating the plug-in module comprises the steps of:

obtaining plug-in initiation information corresponding to the plug-in module definition associated with the identity of the plug-in module;

instantiating the plug-in module based upon a plug-in module definition associated with the identity of the plug-in module; and

passing the plug-in initiation information to the plug-in module for use by the plug-in module.

5. (Original) The method of claim 3 wherein the step of instantiating the plug-in module comprises the step of:

querying a dependency interface associated with the plug-in module with a dependency query to obtain the dependency response from the plug-in module.

6. (Original) The method of claim 1 wherein the step of calculating a plug-in initiation order based upon the dependency list comprises the step of:

arranging a placement of each plug-in module identified in the dependency list within the plug-in initiation order such that plug-in modules not requiring services provided by other plug-in modules are placed earlier in the initiation order and such that plug-in modules requiring services provided by other plug-in modules are placed later in the initiation order.

7. (Original) The method of claim 6 wherein the step of arranging placement of each plug-in module identified in the dependency list within the plug-in initiation order comprises the steps of:

analyzing the dependency list indicating respective plug-in services provided by, and required by, each plug-in module to determine which plug-ins provide services relied upon by other plug-in modules; and

creating, as the plug-in initiation order, at least one plug-in module dependency tree based on the step of analyzing, the at least one plug-in module dependency tree containing a hierarchical arrangement of nodes associated with respective plug-in modules, the hierarchical arrangement indicating the plug-in initiation order of the plug-in modules respectively associated with the nodes in the dependency tree.

8. (Previously Presented) The method of claim 7 wherein initiating service operation of plug-in modules according to the plug-in initiation order comprises:

traversing the at least one plug-in module dependency tree according to the hierarchical arrangement of nodes and for each node encountered during the step of traversing, initiating service operation of the respective plug-in module associated with that node.

9. (Previously Presented) The method of claim 8 wherein the step of initiating service operation of plug-in modules includes:

forwarding, via a dependency available interface associated with a respective plug-in module, a list of initiated plug-in services of other plug-in modules that are currently available for use by the respective plug-in module.

10. (Original) The method of claim 1 wherein the step of initiating service operation of plug-in modules according to the plug-in initiation order comprises performing, for each respective plug-in module in the plug-in initiation order, the steps of:

determining, from a published list of services available by initiated plug-in modules, an identity of each initiated plug-in service required by the respective plug-in module;

forwarding to the respective plug-in module, via a dependency available interface associated with the respective plug-in module, the identity of each initiated plug-in service required by the respective plug-in module;

receiving a list of services initiated by the respective plug-in module; and

adding the list of services provided by the respective plug-in module to the published list of services.

11. (Original) The method of claim 1, wherein the step of initiating service operation of plug-in modules according to the plug-in initiation order operates such that if the second plug-in module requires a service provided by the first plug-in module, the second plug-in module is initiated such that the service provided by the first plug-in module is available to the second plug-in module when required.

12. (Previously Presented) The method of claim 1 wherein the first plug-in module is initiated via the step of initiating service operation of plug-in modules prior to initiation of the second plug-in module.

13. (Original) The method of claim 1 wherein the first plug-in module is initiated via the step of initiating operation of plug-in modules after initiation of the second plug-in module, and wherein the second plug-in module includes a wait-state operation causing the second plug-in module to wait to provide the service offered by the second plug-in module until initiation of the first plug-in module.

14. (Original) The method of claim 1 wherein the steps of obtaining, retrieving, calculating and initiating are performed by a multi-threaded plug-in manager and wherein the step of calculating a plug-in initiation order is performed by collectively operating a respective thread for each plug-in, each thread performing the step of initiating service operation of at least one plug-in module when all services required by that plug-in module are available.

15. (Previously Presented) A computer system comprising:

a memory;

a processor; and

an interconnection mechanism coupling the memory and the processor;

wherein the memory is encoded with a plug-in manager application that, when performed on the processor, produces a plug-in manager process that

manages services associated with a plurality of plug-in modules encoded within the memory by performing the operation steps of:

obtaining identities of a plurality of plug-in modules in the memory;

based on queries to the plurality of plug-in modules, retrieving, into the memory, a dependency list indicating respective plug-in services provided by, and required by, each plug-in module identified in the identities of a plurality of plug-in modules;

calculating, in the memory, a plug-in initiation order based upon the dependency list indicating respective plug-in services provided by, and required by, each plug-in module; and

initiating service operation of plug-in modules on the processor according to the plug-in initiation order, such that if a first plug-in module provides a service required by a second plug-in module, the first plug-in module is initiated such that the service provided by the first plug-in module is available to the second plug-in module when required by the second plug-in module.

16. (Previously Presented) The computer system of claim 15 wherein when the plug-in manager process performs the step of obtaining identities of a plurality of plug-in modules, the plug-in manager process performs the steps of:

receiving a list of services to be started within the computer system;

determining, for each service in the list of services, a respective plug-in module definition that can provide that service; and

placing the identity of each plug-in module definition determined in the step of determining into the identities of the plurality of plug-in modules.

17. (Original) The computer system of claim 15 wherein when the plug-in manager process performs the step of retrieving a dependency list indicating respective plug-in services provided by, and required by, each plug-in module, the plug-in manager process performs the steps of:

for each plug-in module identified in the identities of a plurality of plug-in modules, performing the steps of:

instantiating the plug-in module in the memory based upon a plug-in module definition associated with the identity of the plug-in module;

receiving a dependency response from the plug-in module, the dependency response indicating respective plug-in services provided by, and required by, the plug-in module; and

storing, in the memory, identities of the respective plug-in services provided by, and required by, the plug-in module as identified in the dependency response in the dependency list.

18. (Original) The computer system of claim 17 wherein when the plug-in manager process performs the step of instantiating the plug-in module, the plug-in manager process performs the steps of:

obtaining, in the memory, plug-in initiation information corresponding to the plug-in module definition associated with the identity of the plug-in module;

instantiating the plug-in module in the memory based upon a plug-in module definition associated with the identity of the plug-in module; and

passing the plug-in initiation information to the plug-in module in the memory for use by the plug-in module.

19. (Original) The computer system of claim 17 wherein when the plug-in manager process performs the step of instantiating the plug-in module, the plug-in manager process performs the step of:

querying a dependency interface associated with the plug-in module with a dependency query to obtain the dependency response from the plug-in module.

20. (Original) The computer system of claim 15 wherein when the plug-in manager process performs the step of calculating a plug-in initiation order based upon the dependency list, the plug-in manager process performs the step of:

arranging a placement of each plug-in module identified in the dependency list within the plug-in initiation order such that plug-in modules not requiring services provided by other plug-in modules are placed earlier in the initiation order and such that plug-in modules requiring services provided by other plug-in modules are placed later in the initiation order.

21. (Original) The computer system of claim 20 wherein when the plug-in manager process performs the step of arranging placement of each plug-in module identified in the dependency list within the plug-in initiation order, the plug-in manager process performs the steps of:

analyzing the dependency list indicating respective plug-in services provided by, and required by, each plug-in module to determine which plug-ins provide services relied upon by other plug-in modules; and

creating in the memory, as the plug-in initiation order, at least one plug-in module dependency tree based on the step of analyzing, the at least one plug-in module dependency tree containing a hierarchical arrangement of nodes associated with respective plug-in modules, the hierarchical arrangement indicating the plug-in initiation order of the plug-in modules respectively associated with the nodes in the dependency tree.

22. (Original) The computer system of claim 21 wherein when the plug-in manager process performs the step of initiating service operation of plug-in modules according to the plug-in initiation order, the plug-in manager process performs the step of:

traversing the at least one plug-in module dependency tree in the memory according to the hierarchical arrangement of nodes and for each node encountered during the step of traversing, initiating service operation of the respective plug-in module associated with that node.

23. (Original) The computer system of claim 22 wherein when the plug-in manager process performs the step of initiating service operation of the respective plug-in module associated with that node, the plug-in manager process performs the step of:

forwarding, via a dependency available interface associated with the respective plug-in module, a list of initiated plug-in services of other plug-in modules that are currently available for use by the respective plug-in module.

24. (Original) The computer system of claim 15 wherein when the plug-in manager process performs the step of initiating service operation of plug-in modules

according to the plug-in initiation order the plug-in manager process performs, for each respective plug-in module in the plug-in initiation order, the steps of:

determining, from a published list of services available by initiated plug-in modules, an identity of each initiated plug-in service required by the respective plug-in module;

forwarding to the respective plug-in module, via a dependency available interface associated with the respective plug-in module, the identity of each initiated plug-in service required by the respective plug-in module;

receiving a list of services initiated by the respective plug-in module; and

adding the list services provided by the respective plug-in module to the published list of services.

25. (Original) The computer system of claim 15, wherein the step of initiating service operation of plug-in modules according to the plug-in initiation order operates in the plug-in manager process such that if the second plug-in module requires a service provided by the first plug-in module, the second plug-in module is initiated such that the service provided by the first plug-in module is available to the second plug-in module when required.

26. (Previously Presented) The computer system of claim 15 wherein the plug-in manager initiates the first plug-in module via the step of initiating service operation of plug-in modules prior to initiation of the second plug-in module.

27. (Original) The computer system of claim 15 wherein the plug-in manager process initiates the first plug-in module via the step of initiating operation of plug-in modules after initiation of the second plug-in module, and wherein the second plug-in module includes a wait-state operation causing the second plug-in module to wait to provide the service offered by the second plug-in module until initiation of the first plug-in module.

28. (Original) The computer system of claim 15 wherein the steps of obtaining, retrieving, calculating and initiating are performed by a multi-threaded plug-in manager and wherein the step of calculating a plug-in initiation order is performed

by collectively operating a respective thread for each plug-in, each thread performing the step of initiating service operation of at least one plug-in module when all services required by that plug-in module are available.

29. (Previously Presented) A computer program product having a computer-readable medium including computer program logic encoded thereon, that when executed on a computer system having a coupling of a memory and a processor, provides a plug-in manager process for managing plug-in services by causing the processor to perform the operations of:

obtaining identities of a plurality of plug-in modules in the memory;

retrieving, into the memory, a dependency list indicating respective plug-in services provided by, and required by, each plug-in module identified in the identities of a plurality of plug-in modules;

calculating, in the memory, a plug-in initiation order based upon the dependency list indicating respective plug-in services provided by, and required by, each plug-in module;

initiating service operation of plug-in modules on the processor according to the plug-in initiation order, such that if a first plug-in module provides a service required by a second plug-in module, the first plug-in module is initiated such that the service provided by the first plug-in module is available to the second plug-in module when required by the second plug-in module; and

querying a dependency interface associated with the plug-in module with a dependency query to obtain a dependency response from the plug-in module, the dependency response indicating respective plug-in services provided by the plug-in module.

30. (Previously Presented) A computer system comprising:

a memory;

a processor; and

an interconnection mechanism coupling the memory and the processor;

wherein the memory is encoded with a plug-in manager application that, when performed on the processor, produces a plug-in manager process that manages services associated with a plurality of plug-in modules encoded within the

memory by operating on the computer system and causing the computer system to provide:

means for obtaining identities of a plurality of plug-in modules in the memory;

means for retrieving, into the memory, a dependency list indicating respective plug-in services provided by, and required by, each plug-in module identified in the identities of a plurality of plug-in modules;

means for calculating, in the memory, a plug-in initiation order based upon the dependency list indicating respective plug-in services provided by, and required by, each plug-in module;

means for initiating service operation of plug-in modules on the processor according to the plug-in initiation order, such that if a first plug-in module provides a service required by a second plug-in module, the first plug-in module is initiated such that the service provided by the first plug-in module is available to the second plug-in module when required by the second plug-in module; and

wherein the first plug-in module is initiated via the step of initiating operation of plug-in modules after initiation of the second plug-in module, and wherein the second plug-in module includes a wait-state operation causing the second plug-in module to wait to provide the service offered by the second plug-in module until initiation of the first plug-in module.

31. (Previously Presented) A computer program product as in claim 29, wherein the processor further performs operations of:

determining a list of plug-in services required by a software application; and

querying a set of plug-in modules to identify services provided by the set of plug-in modules.

32. (Previously Presented) A computer program product as in claim 31, wherein the processor further performs operations of:

in response to querying the set of plug-in modules, identifying plug-in modules not identified by the software application as being necessary but which are identified by the set of plug-in modules as being necessary to carry out execution of an operation on behalf of the software application.

33. (Previously Presented) A computer program product as in claim 32, wherein the processor further performs operations of:
initiating service operation of plug-in modules on the processor according to an order other than the plug-in initiation order, such that if a third plug-in module provides a service required by a fourth plug-in module, the third plug-in module being initiated after initiation of the fourth plug-in module, the third plug-in module initiating a wait state operation causing the third plug-in module to wait to provide the service offered by the third plug-in module until instantiation of the fourth plug-in module.

34. (Previously Presented) A computer program product as in claim 32, wherein the processor initiates execution of the first plug-in module before execution of the second plug-in module, the first plug-in module initiating a wait state operation resulting in signaling to the second plug-in module, the signaling indicating that a respective service of the first plug-in module is not yet available to the second plug-in module.

35. (Previously Presented) A computer program product as in claim 34, wherein the processor further performs operations of:
maintaining a list of services for a set of plug-in modules currently able to provide respective services; and
publishing the list of services for the software application to identify instantiated plug-ins currently providing the respective services.